# Introdução

Em diversas aplicações corporativas nos deparamos ao analisar as suas bases de dados com relacionamentos *um-para-muitos* entre duas entidades. Exemplos são fáceis de encontrar. Em um sistema como o SIGA/UFRJ, a encontramos na relação entre um *Curso* e a sua lista de *Alunos* ativos. Em um sistema de gerenciamento hospitalar, entre um *Médico* e a lista de *Pacientes* sob seu atendimento em um determinado dia, os quais, por sua vez, podem ter uma lista de *Exames* associados. Outro exemplo bem comum ocorre em aplicações de e-commerce, como em sites de vendas on-line, onde temos a lista de *Pedidos* feitas por um dado *Cliente*, onde cada *Pedido*, por sua vez, pode ter uma lista de *Itens*, com informações de descrição, quantidades e valores. Em redes sociais encontramos o exemplo da lista de *Mensagens* recebidas por um certo *Usuário*, ou a lista de *Contatos* desse mesmo *Usuário*.

Em cada caso citado, apesar da modelagem do relacionamento ser sempre *um-para-vários*, as características intrínsecas de cada aplicação podem requerer o uso de métodos distintos de acesso aos dados envolvidos no relacionamento, juntamente com a Estrutura de Dados subjacente, ou seja, esse tipo de relacionamento pode ser implementado, ou codificado, de diversas formas, utilizando tipos distintos de Estruturas de Dados, como um conjunto específico de algoritmos que, juntos, dão solução aos requisitos da aplicação. Uma das estruturas de dados mais utilizada para esse fim são as *listas* encadeadas, simples ou duplas, ou mesmo circulares, ordenadas ou não.

Nos exemplos citados, no caso da lista de Alunos de um Curso, uma lista simples, não ordenada é o suficiente, sendo que as operações de Inserção de novo Aluno e Remoção dos Alunos já formados constituem as principais *operações de manipulação* com a “Lista de Alunos de um Curso”. Adicionalmente, saber a que curso pertence um dado Aluno, e a listagem dos alunos ativos em um dado Curso, são as principais *operações de consulta* à essa lista.

Nesse primeiro trabalho de avaliação, que pode ser feito em grupo de do máximo cinco alunos, o objetivo é usar listas encadeadas, simples ou duplas, para codificar justamente esse tipo de relacionamento *um-para-muitos*. A intensão subjacente é avaliar o aprendizado do básico de programação em C++, englobando a construção de hierarquias de classes orientado a objetos, e, principalmente, o **domínio da programação de listas encadeadas**.

Iremos também aproveitar esse primeiro trabalho para introduzir algumas técnicas básicas de Engenharia de Software, área do conhecimento que surgiu da interseção entre a Engenharia de Produção e a Ciência da Computação, voltada para a criação, desenvolvimento e manutenção de software em geral, seja ele um sistema operacional tal como o Windows®, ou um jogo como o League of Legends®, ou um APP de celular como o Whatsapp®. Durante esse “ciclo de vida do produto software”, são utilizadas diversas tecnologias da Ciência da Computação e práticas de gerência de projetos, visando a organização e otimização do uso dos recursos empregados no desenvolvimento, a produtividade nesse desenvolvimento, tudo isso com o devido controle da *qualidade* do processo todo e do produto final: o software.

# Especificação do Trabalho

Escolher um recorte mínimo de alguma aplicação que possua base de dados com relacionamentos *um-para-muitos*, como os citados na introdução. Por exemplo, o recorte que codificamos no Exemplo 003, feito em sala de aula, da aplicação SIGA/UFRJ, com apenas três entidades: Curso, Aluno e Professor. Na modelagem de dados da mini aplicação, procure introduzir algum relacionamento de *especialização – generalização*, a fim de codificar alguma Hierarquia de Classes OO, como fizemos no citado Exemplo 003, quando criamos a Classe Abstrata Pessoa, que generalizava as Classe Aluno e Professor.

## Análise do Sistema:

Após a escolha da “mini aplicação” a ser desenvolvida, a primeira etapa do “ciclo de vida” da produção de um software a ser feita é a “Análise do Sistema”. Nela procure descrever, o mais pormenorizadamente possível, o “problema” ao qual o software se propõe a resolver. Simplificando ao extremo essa etapa, o grupo deverá apenas produzir um “texto livre” descrevendo todos os possíveis detalhes e funcionalidades que o sistema deve oferecer ao seu usuário. Ao final desse texto deverá haver uma “Lista de Requisitos” do sistema a ser desenvolvido. Por exemplo, tomando novamente o Exemplo 003, a lista seria:

**Lista de Requisitos do Mini-SIGA:**

1.       Gerenciar o Cadastro de Cursos, Professores e Alunos. Isso pressupõe operações de:

a.       Cadastro de novo Curso, Professores ou Aluno

b.       Consulta, por qualquer dos seus atributos, de Curso, Professores e Aluno

c.       Listagem dos cadastros

d.       Alteração de cadastrados

e.       Remoção de cadastrados

2.       Curso possuem as informações (atributos) de “Nome do Curso” e “Código do Curso”, Cada Curso cadastrado possui uma lista de seus alunos ativos. Assim, a cada semestre, há a inclusão dos calouros do curso nessa lista e a remoção dos alunos formandos.

3.       Cursos podem listar os nomes e DRE dos seus alunos ativos.

4.       Professores possuem a informação de CPF, Nome, número SIAPE, uma lista de Alunos Orientados. O máximo de alunos que um professor pode orientar de cada vez é 10.

5.       Professores podem procuras as informações de um dado aluno por ele orientado, bem como obter a lista dos nomes e DRE dos seus orientados.

6.       Alunos possuem a informação de CPF, Nome, número DRE e CR.

7.       Alunos podem consultar a que Curso pertencem e qual o nome do seu professore orientador.

## Projeto do Sistema:

A etapa seguinte a Análise vem a ser o “Projeto do Sistema”. Nesta é onde se dá a descrição o mais completa possível da “solução computacional ao problema” apresentado e especificado na fase de Análise do Sistema. O Projeto do sistema irá nortear as fases seguintes de Codificação, Testes e Homologação do Sistema (aceite por parte do cliente, neste caso, o professor). Para tornar simples esta etapa, o grupo deverá elaborar apenas dois diagramas com suas respectivas tabelas descritivas. O primeiro diagrama deverá apresentar o “Modelo de Dados” do sistema, o qual será reduzido a apenas um “Diagrama de Classes” contendo a figura da Hierarquia de Classe OO do projeto. O segundo diagrama deverá ser um “Modelo Funcional” do sistema, o qual será reduzido a apenas um “Diagrama de Módulos” do sistema, contendo a figura da árvore de dependência entre os módulos do sistema, cada qual com a sua lista de Funções.

Para a solução proposta no projeto a **única restrição** que o trabalho terá é que a solução para a implementação de *relacionamentos um-para-muitos*, especificados no modelo de dados do sistema, deverá ser com a utilização de listas encadeadas totalmente desenvolvidas no trabalho, ou seja, não se pode usar a classe *list* (ou nenhum outro container) da biblioteca padrão STL do C++ na solução desses relacionamentos. Porém é livre o uso de qualquer biblioteca C++, padrão ou não, para a implementação dos demais componentes do sistema.

Para cada classe do “Diagrama de Classes”, deverá haver uma Tabela de Descrição da Classe, onde serão colocados todos os detalhes necessários à sua codificação, tais como:

1.       Nome e descrição geral da classe,

2.       Se é uma classe abstrata,

3.       Quais são as suas superclasses (generalizações)

4.       A lista de subclasses derivadas (especializações),

5.       A lista de *atributos de dados e funções* (métodos) herdados, bem como o tipo de herança em cada caso,

6.       A lista de *atributos* *de dado*, sendo que para cada um desses atributos deverá ser especificado o *nome do atributo*, a sua descrição, o *domínio* (tipo ou classe, p. exe., inteiro, ponto flutuante, string etc.), se são atributos de dados *de* *objeto* ou *de classe*, suas restrições de integridade, tais como, valores de inicialização (*default values*), se são constantes ou não, se são campos chave (valores únicos e não nulos), ou se admite valores repetidos, se admite valores nulos ou desconhecidos (*null values*, *unknow values*), faixa de valores (p.exe. idades de 0 a 60 anos), ou se são campos *chave estrangeira* (*lookup field*, ou seja, o seu valor deve estar entre um dos valores atuais de um certo atributo de dado de uma coleção de objetos, ou de uma tabela de *lookup*), e, por fim, observações pertinentes ao atributo.

7.       A lista de *atributos funções* (métodos do objeto), sendo que para cada um deles deverá ser especificado o nome da função, a sua descrição geral, *suas complexidades média e do pior caso*, se é função membro *de objeto* ou *de classe* (*storage level*), o seu nível de encapsulamento (visibilidade, ou escopo), se é uma função da interface do objeto ou é interna ao objeto, se são funções virtuais (*linkage level*), ou seja, se deverão ser necessariamente sobrecarregada (*overloading*) por sua subclasses derivadas, a descrição da sua lista de parâmetros, com tipo/classe de cada parâmetro e seu tipo de passagem (in, out, in/out, ou passagem por valor ou por referência), o tipo/classe do valor de retorno, se levanta exceções e quais, a lista de *atributos de dados* da classe que a função afeta, e como (ou seja, se realiza operação de *insert, selec/retrieval t, update ou delete*), a lista de objetos (estruturas de dados) externos a classe que afeta, e, por fim, quando pertinente, o pseudocódigo da função.

Para cada módulo do “Diagrama de Módulos”, deverá haver uma Tabela de Descrição do Módulo, onde serão especificados todos os detalhes necessários à sua codificação, tais como:

1.       Nome e descrição geral do módulo.

2.       Função geral do módulo, ou seja, se está implementando processamentos do modelo de dados da aplicação, ou processamentos de visualização de dados ou da interface com o usuário, ou se implementa regras de controle ou de fluxo da aplicação.

3.       A lista de submódulos que o compõe

4.       A lista de *objetos e/ou coleções* (estruturas de dados da aplicação) que utiliza e afeta, e como, entre operação de *create*, *insert, select/retrieval, update ou delete*, informando se são internos ao módulo ou externos a este (globais ou de outros módulos).

5.       A lista de *funções* (métodos globais do módulo), sendo que para cada um deles deverá ser especificado o nome da função, a sua descrição geral, *suas complexidades média e do pior caso*, se é função interna do módulo ou da sua interface (*storage level*), a visibilidade e escopo da função, a descrição da sua lista de parâmetros, com tipo/classe de cada parâmetro e seu tipo de passagem (in, out, in/out, ou passagem por valor ou por referência), o tipo/classe do valor de retorno, se levanta exceções e quais, a lista de objetos e coleçõesque a função afeta, e como (operação de *create*, *insert, select/retrieval, update ou delete*), a lista de objetos externos ao módulo que afeta, e, por fim, quando pertinente, o pseudocódigo da função.

6.       A lista de módulos, funções e objetos externos que acionam funções e interagem com objetos do módulo.

# Regras do Desenvolvimento do Trabalho

1.       Grupos de no máximo cinco alunos

2.       Fazer e entregar três documentos: o primeiro é a “Especificação de Requisitos do Sistemas”, como descrito na fase de Análise do Sistema. O segundo é o Projeto do Sistema, composto de Diagrama de Classes e Diagrama de Módulos, ambos entregue juntos com as tabelas descritivas como descrito acima. O terceiro documento apresenta os testes realizados como o sistema e as conclusões gerais sobre o trabalho.

3.       Toda a codificação deverá ser feita em linguagem C++, utilizando qualquer compilador e IDE de escolha do grupo, bem como a sua biblioteca padrão básica, STL ou outra qualquer.

4.       Toda a documentação do projeto e o código desenvolvido deverá ser entregue através do GitHub do projeto

5.       Em resposta à essa tarefa cada componente do grupo deverá postar um folha com a lista de componentes do grupo, como nome completo e DRE, e o link para a página do GitHub do projeto.

6.       Prazo de entrega: duas semanas